CHALMERS
UNIVERSITY OF TECHNOLOGY

Learning Negotiating Behavior Between Cars in Intersections using Deep Q-Learning

Learning Negotiating Behavior Between Cars in Intersections using Deep Q-Learning

Tommy Tram[1,2], Anton Jansson[1], Robin Grönberg[1], Mohammad Ali[1], Jonas Sjöberg[2]

***Abstract*— This paper concerns automated vehicles negotiating with other vehicles, typically human driven, in crossings with the goal to find a decision algorithm by learning typical behaviors of other vehicles. The vehicle observes distance and speed of vehicles on the intersecting road and use a policy that adapts its speed along its pre-defined trajectory to pass the crossing efficiently. Deep Q-learning is used on simulated traffic with different predefined driver behaviors and intentions. The results show a policy that is able to cross the intersection avoiding collision with other vehicles 98% of the time, while at the same time not being too passive. Moreover, inferring information over time is important to distinguish between different intentions and is shown by comparing the collision rate between a Deep Recurrent Q-Network at 0.85% and a Deep Q-learning at 1.75%.**

I. Introduction

The development of autonomous driving vehicles is fast and there are regularly news and demonstrations of impressive technological progress [1]. However, one of the largest challenges does not have to do with the autonomous vehicle itself but rather their interaction with human driven vehicles in mixed traffic situations. Human drivers are expected to follow traffic rules strictly, but in addition they also interact with each other in a way which is not captured by the traffic rules [2], [3]. This *informal* traffic behavior is important, since the traffic rules alone may not always be enough to give the safest behavior. This motivates the development of control algorithms for autonomous vehicles which behave in a "human-like" way, and in this paper we investigate the possibilities to develop such behavior by training on simulated vehicles.

In [4] they raise two concerns when using Machine learning, specially Reinforcement learning, for autonomous driving applications: ensuring functional safety of the Driving Policy and that the Markov Decision Process model is problematic, because of unpredictable behavior of other drivers. In the real world, intentions of other drivers are not always deterministic or predefined. Depending on their intention, different actions can be chosen to give the most comfortable and safe passage through an intersection. They also noted that in the context of autonomous driving, the dynamics of vehicles is Markovian but the behaviors of other road users may not necessarily be Markovian.

These two concerns are addressed using a Partially Observable Markov Decision Process (POMDP) as a model and Short Term Goals (STG) as actions. With a POMDP the unknown intentions can be estimated using observations and that has shown promising results for other driving scenarios [5]. The POMDP is solved using a model-free approach called Deep (Recurrent) Q-Learning. An initial study of this approach was performed in [6] and in this paper we show that the policy is able to learn a negotiating behavior without knowing other drivers' intentions. With this approach a driving policy can be found using only observations without defining the MDP states. Since we do not train on human driven vehicles, the results presented here cannot be considered human-like, but the general approach, to train the algorithms using traffic data, is shown working, and a possible next step could be to start with the pre-tuned policies from this work, and to continue the training in real traffic crossings.

II. Overview

This paper starts by introducing the system architecture and defining the actions, observations and POMDP in Section III. The final strategy of what action to take at a given situation is called a policy and is described in Section IV. Deep Q-learning is used to find this policy, which uses a neural network to approximate a Q-value and is described in Section V together with techniques used to improve the learning, such as Experience replay, Dropout and a recurrent layer called Long Short-Term Memory (LSTM). We then present the simulation, reward function and neural network configurations in Section VI. The results are then presented in Section VII comparing the effect of the methods mentioned in Section V. Finally, the conclusion and brief discussion is presented in Section VIII.

III. Problem formulation

The objective is to drive along a main road that has one or two intersections with crossing traffic and control the acceleration in a way that avoids collisions in a comfortable way. All vehicles are assumed to drive along predefined paths on the road where they can either speed up or slow down to avoid collisions in the crossings. In this section the system architecture is defined along with the environment, observations and actions.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

[1]Tommy Tram, Anton Jansson, Robin Grönberg and Mohammad Ali are with Zenuity AB, Gothenberg, Sweden {tommy.tram, anton.jansson, robin.gronberg, mohammad.ali}@zenuity.com

[2]Tommy Tram and Jonas Sjöberg are with the Department of Electrical Engineering, Chalmers University of Technology, Gothenberg, Sweden {tram, jonas.sjoberg}@chalmers.se

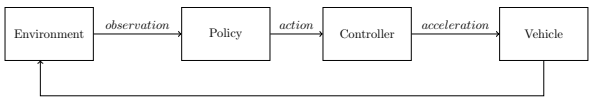

Fig. 1. Representation of the architecture

A. System architecture

Environment is defined as the world around the ego vehicle, including all vehicles of interest and the shape/type of the intersection. The environment can vary in different ways, e.g. number of vehicles and intersections or the distance to intersections. The environment is defined by the simulation explained in section VI-A. We assume that the ego vehicle receives observations from this environment at each sampling instant, as shown in Fig. 1. A policy then takes these observations and chooses a high level action that is defined in more detail in section III-C. These actions are sent to a controller that calculates the appropriate acceleration request given to the ego vehicle, which will influence the environment and impact how other cars behave.

B. Actions as Short Term Goals

Motivated by the insight that the ego vehicle has to drive before or after other vehicles when passing the intersection, decisions on the velocity profile is modeled by simply keeping a distance to other vehicles until they pass. This is done by defining the actions as Short Term Goals (STG), eg. keep set speed or yield for crossing car. This allows the properties of comfort on actuation and safety to be tuned separately, reducing the policy selection to a classification problem. The actions are then as follows:

- *Keep set speed*: Aims to keep a specified maximum speed $v_{\max}$, using a simple P-controller

$$a_p^e = K(v_{max} - v^e) \tag{1}$$

where a_p^e is the acceleration request and v^e is the velocity of ego vehicle towards the center of the intersection, while K is a proportional constant.

- *Keep distance to vehicle* N: Will control the acceleration in a way that keeps a minimum distance to a chosen vehicle N, a *Target Vehicle*, and can be implemented using a sliding mode controller, where the acceleration request is computed as

$$a_{sm}^e = \frac{1}{c_2}(-c_1 x_2 + \mu sign(\sigma(x_1, x_2))) \tag{2}$$

$$\text{where} \begin{cases} x_1 = p^t - p^e \\ x_2 = v^t - v^e \end{cases}$$

where p^e and p^t is the position of ego and target vehicle respectively, shown in Fig. 2, and v^t is the velocity of target vehicle. c_1 together with c_2 are calibration parameters that can be set to achieve wanted performance with a surface

$$\sigma = c_1 x_1 + c_2 x_2 \tag{3}$$

The final acceleration request is then achieved by

$$a^e = \min(a_{sm}^e, a_p^e) \tag{4}$$

For more detailed information about sliding mode see [7]. To distinguish between different cars to follow, each vehicle will have its own action.

- *Stop in front of intersection*: Stops the car at the next intersection. Using the same controller as eq. 4 while setting $v^t = 0$ and p^t to start of intersection, the controller can bring ego vehicle to a comfortable stop before the intersection.

C. Observations that make up the state

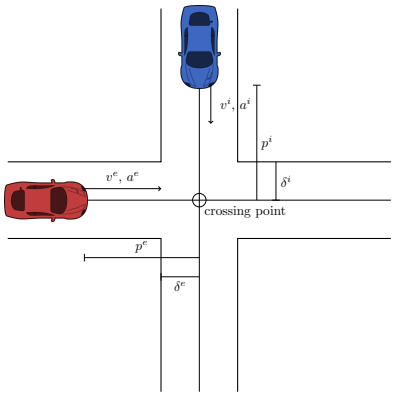

Fig. 2. Observations that makes the state

A human driver is, generally, good at assessing a scenario and it is hard to pin-point what information is used in their assessment. Therefore some assumptions are made on which features that are interesting to observe. The observation o_t at time t is defined as:

$$o_t = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^i \quad v_t^i \quad a_t^i \quad \delta^i \quad a_{t+1}^{e,A}\ ]^T \tag{5}$$

With notations as follows: consider Fig. 2, position of ego p_t^e and other vehicle p_t^i are defined as distance to common reference point, called *crossing point*, where i is an index of the other vehicle. The start of intersection for ego δ^e and other vehicle δ^i also uses the crossing point as reference. These are relevant in case a driver would choose to yield for other vehicles, then they would most likely stop before the start of intersection. The velocity v^e and acceleration a^e of ego vehicle and velocity v^i and acceleration a^i of the other vehicles are observed to include the dynamics of different actors. The last feature in the observation, $a_{t+1}^{e,A}$, is the ego vehicle's predicted acceleration for each possible action A, which can be used to account for comfort in the decision.

D. Partially Observable Markov Decision Processes

The decision making process in the intersection is modeled as a POMDP. A POMDP works like a Markov Decision Process (MDP) [8] in most aspects, but the full state is not observable.

At each time instant, an action, $a_t \in \mathcal{A}$, is taken, which will change the environment state s_t to a new state vector

s_{t+1}. Each action a_t from a state s_t has a value called the reward r_t, which is given by a reward function $\mathcal{R}_t$.

One of the unobservable states could be the intentions of other drivers approaching the intersection. The state can only be perceived partially through observations $o_t \in \Omega$ with the probability distribution of receiving observation o_t given an underlying hidden state $s_t : o_t \leftarrow \mathcal{O}(s_t)$, where $\mathcal{O}(s_t)$ is the probability distribution.

IV. Finding the optimal policy

Assuming the MDP states are not known, we want a model-free method of finding a policy, and for this we use reinforcement learning. The goal is to have an agent learn how to maximize the future reward by taking different actions in a simulated environment. Details on the simulation environment used is described in Section VI-A. The strategy of which action to take given a state is called a policy π and can be modeled in two ways:

- As a stochastic policy $\pi(a|s) = \mathcal{P}[\mathcal{A} = a|\mathcal{S} = s]$
- As a deterministic policy $a = \pi(s)$

The standard assumption is made that the future reward is discounted by a factor γ per time step, making the discounted future reward $\mathcal{R}_t = \sum_t^\tau \gamma^{t-1} r_t$, where τ is the time step where the simulation ends, e.g. when the agent crosses an intersection safely.

Similar to [9], the optimal action-value function $Q^*(s_t, a_t)$ is defined as the maximum expected reward achievable by following a policy π given the state s_t and taking an action a_t:

$$Q^*(s_t, a_t) = \max_\pi \mathbb{E}[\mathcal{R}_t|s_t, a_t, \pi] \tag{6}$$

Using the Bellman equation, $Q^*(s_t, a_t)$ can be defined recursively. If we know $Q^*(s_t, a)$ for all actions a that can be taken in state s_t, then the optimal policy will be one that takes the action a_t that gives the highest immediate and discounted expected future reward $r_t + \gamma Q^*(s_{t+1}, a_{t+1})$. This gives us:

$$Q^*(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})|s_t, a_t] \tag{7}$$

The optimal policy π^* is then given by taking actions according to an optimal $Q^*(s_t, a_t)$ function:

$$\pi^*(s_t) = \arg\max_{a_t} Q^*(s_t, a_t) \tag{8}$$

V. Method

In this section we will briefly describe Q-learning and methods used to improve the learning such as, Experience replay, dropout and Long Short-Term Memory.

A. Deep Q-learning

From eq. 8, the optimal policy is defined by taking an action that has the highest expected Q-value. Because the Q-value is not known, a non linear function approximation, such as a neural network, is used to estimate the Q-function. The method is known as Deep Q-Learning [9]. The neural network used to approximate the Q-function is called a Deep Q-network (DQN) and is denoted as $Q(s_t, a_t|\theta^\pi)$, where θ^π is the weight and biases of the neural network for a policy π. The state s_t is the input to the DQN and the output is the Q-value for each action $a_t \in \mathcal{A}$.

B. Experience Replay

Experience replay, as proposed by [9], is a method that stores all observations o, together with taken actions a and their rewards r as an experience memory E and then trains the DQN using sample from experiences E'.

Looking at eq. 8, the optimal policy is greatly affected by Q-function and if the DQN is only trained on recent experiences E, the distribution will have a bias towards recent experiences. As shown by [10], this can give undesired effects on the feedback loops and lead to divergence of the DQN.

By using experience replay, the network is instead trained on the average of the experience. Thus reducing the time for convergence of the DQN and oscillation due to training on the same experience multiple times [11].

C. Dropout

Overfitted neural networks have bad generalization performance [12] and to help reduce overfitting a technique called dropout was used. By temporarily removing some hidden neurons with probability p in the network before each training iteration, the network learns to adapt and generalize instead of depending too strongly on a few hidden neurons. For more details, see [13].

D. Long short-term memory

The effect of observed behaviors over time is explored in this paper and is done by adding a recurrent layer to the DQN making it a Deep Recurrent Q-Network (DRQN). A regular recurrent layer has difficulties with longer sequences because of vanishing gradients, and [14] showed that using an LSTM solves this problem. Instead of storing all information from the previous time sample, LSTM stores information in a memory cell and modifies it by using insert and forget gates. These gates decide if a memory cell should be kept or cleared and is learned by the network. This enables both recent and older observations to be stored and utilized by the network. A sequence length of 4 is used when training the LSTM, where the first 3 observations are only used to build the internal memory state of the LSTM cells, as described in [15].

VI. Implementation

In this section we go through the experiment implementation. A simulation environment was set up to model the interactions. From section III, both the number of observations and actions are dependent on the maximum number of cars. In this paper we consider up to 4 cars. The Deep Q Network can then also be fully defined with the help of observations from section III and finally we go through the reward function that defines our behavior.

A. Simulation environment

The simulation environment is set up as an intersection described in section III-C. The number of other cars that are observable at the same time can vary from 1-4, while their intentions can vary between an aggressive *take way*, passive *give way* or a cautious driver. The take way driver does not slow down or yield for crossing traffic in an intersection, while the give way driver will always yield for other vehicles before continuing through the intersection. The cautious driver on the other hand, will slow down for crossing traffic but not come down to a full stop. With a maximum number of other cars set to 4 all possible actions the ego vehicle can take are:

- α_1: Keep set speed.
- α_2: Stop in front of intersection.
- α_3: Keep distance to vehicle 1.
- α_4: Keep distance to vehicle 2.
- α_5: Keep distance to vehicle 3.
- α_6: Keep distance to vehicle 4.

At the start of an episode, the ego vehicle's position and velocity, the number of other vehicles and their intentions are randomly generated. The episode only ends when the ego vehicle fulfills one out of three conditions: 1) Crossing the intersection and reaching the other side, 2) Colliding with another vehicle. or 3) Running out of time τ_m. Each car follows the control law from eq. 4, trying to keep a set speed while keeping a set distance to the vehicle in front of its own lane. All cars including the ego vehicle in these scenarios have a maximum acceleration set to $5m/s^2$, this was set based on comfort and normal driving conditions.

B. Reward function tuning

Defining the reward function, the distribution was kept around $[-1, 1]$. Large reward values would give large Q_π-values, so the values are kept small to keep the gradients from growing too large [16]. The reward function is defined as follows:

$$r_t = \hat{r}_t + \begin{cases} 1 - \frac{\tau}{\tau_m} & \text{on success,} \\ -2 & \text{on collision} \\ -0.1 & \text{on timeout, i.e. } \tau \geq \tau_m \\ -\left(\frac{j_t^e}{j_{\max}}\right)^2 \frac{\Delta\tau}{\tau_m} & \text{on non-terminating updates} \end{cases}$$

$$\text{where } \hat{r}_t = \begin{cases} -1 & \text{if chosen } a_t \text{ is not valid} \\ 0 & \text{otherwise} \end{cases}$$

The actions $\alpha_3, \ldots, \alpha_6$ described should only be selected when a vehicle is observable and has not crossed the intersection. This is enforced by punishing the agent with a large negative reward $\hat{r}_t$ if an invalid action was selected. Switching between different STG at a high frequency could result in an uncomfortable experience due to high jerk in acceleration. Therefore the agent is also punished for large acceleration jerk j_t^e, where τ is the elapsed time since the episode started, $\Delta\tau$ the time between samples and τ_m is the maximum time before a timeout.

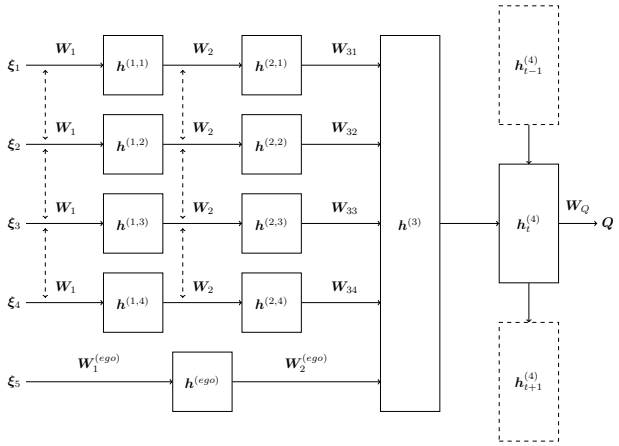

Fig. 3. Deep Recurrent Q Network layout with shared weights and a LSTM

C. Neural Network Setup

The DRQN structure is defined in Fig. 3. Where $\boldsymbol{h}$ are the hidden layers of the network with weights $\boldsymbol{W}$. Because the observations o_t from section 2, are used as input to the DRQN, the number of features must be fixed. With up to four other cars, the input vectors $\boldsymbol{\xi}$ are as follows:

- $\xi_1 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^1 \quad v_t^1 \quad a_t^1 \quad \delta^1\ ]^T$
- $\xi_2 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^2 \quad v_t^2 \quad a_t^2 \quad \delta^2\ ]^T$
- $\xi_3 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^3 \quad v_t^3 \quad a_t^3 \quad \delta^3\ ]^T$
- $\xi_4 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^4 \quad v_t^4 \quad a_t^4 \quad \delta^4\ ]^T$
- $\xi_5 = [\ a_{t+1}^{e,1} \quad a_{t+1}^{e,2} \quad a_{t+1}^{e,3} \quad a_{t+1}^{e,4} \quad a_{t+1}^{e,5} \quad a_{t+1}^{e,6}\ ]^T$

In case a vehicle is not visible, the input vector $\boldsymbol{\xi}$ is set to $-\mathbf{1}$, where $-\mathbf{1}$ is a vector of appropriate dimensions with all elements set to one. The maximum speed $v_{\max}$, maximum acceleration $a_{\max}$ and a car's sight range $p_{\max}$ was used to scale all features down to values between $[-1, 1]$.

The output $\boldsymbol{Q}$ should be independent of which order other vehicles was observed in the input ξ_i. In other words, whether a vehicle was fed into ξ_1 or into ξ_4, the network should optimally result in the same decision, only based on the features' values. The network is therefore structured such that input features of one car, for instance ξ_1, are used as input to a sub-network with two layers $\boldsymbol{h}^{(1,i)}$ and $\boldsymbol{h}^{(2,i)}$. Each other vehicle has a copy of this sub-network, resulting in them sharing weights ($\boldsymbol{W}_1$ and $\boldsymbol{W}_2$), as shown in Fig. 3. The first hidden layers are then given by:

$$\boldsymbol{h}^{(1,i)} = \tanh\left(\boldsymbol{W}_1 \boldsymbol{\xi}_i + \boldsymbol{b}_1\right) \tag{9}$$

$$\boldsymbol{h}^{(2,i)} = \tanh\left(\boldsymbol{W}_2 \boldsymbol{h}^{(1,i)} + \boldsymbol{b}_2\right) \tag{10}$$

$$\boldsymbol{h}^{(ego)} = \tanh\left(\boldsymbol{W}_1^{(ego)} \boldsymbol{\xi}_5 + \boldsymbol{b}^{(ego)}\right) \tag{11}$$

The output of each sub-network, $\boldsymbol{h}^{(2,i)}$ and $\boldsymbol{h}^{(ego)}$, is fed as input into a third hidden layer $\boldsymbol{h}^{(3)}$. The different sub-networks' $\boldsymbol{h}^{(2,i)}$ outputs are multiplied with different weights $\boldsymbol{W}_{31}, \ldots, \boldsymbol{W}_{34}$ in order to distinguish different cars for different follow car actions. The ego features are also fed

into layer 3 with its own weights $\boldsymbol{W}_2^{(ego)}$. The neurons in layer $\boldsymbol{h}^{(3)}$ combine the inputs by adding them together:

$$\boldsymbol{h}^{(3)} = \tanh\left(\boldsymbol{W}_2^{(ego)}\,\boldsymbol{h}^{(ego)} + \sum_{i=1}^{4} \boldsymbol{W}_{3i}\,\boldsymbol{h}^{(2,i)} + \boldsymbol{b}_3\right) \tag{12}$$

The final layer $\boldsymbol{h}^{(4)}$ uses the LSTM, described in section V. This layer handles the storage and usage of previous observations, making it the recurrent layer of the network.

$$\boldsymbol{h}_t^{(4)} = \text{LSTM}\left(\boldsymbol{h}^{(3)} | \boldsymbol{h}_{t-1}^{(4)}\right) \tag{13}$$

The approximated $\boldsymbol{Q}$-value is then

$$\boldsymbol{Q} = \boldsymbol{W}_Q \boldsymbol{h}^{(4)} + \boldsymbol{b}_4 \tag{14}$$

VII. Results

Metrics used to evaluate the performance was mainly the success rate followed by collision to timeout ratio (CTR) and average episodic reward. Success rate is defined as number of times the agent reached the end of path without colliding or reaching the time limit. Because both timeouts and collisions are defined as failures, a CTR was used to distinguish the timeouts from collisions for the last 100 episodes where a high value corresponds to more crashes than timeouts. A collision rate corresponding to the total amount of episodes resulting in a collisions is then computed using success rate and CTR averaged. To compare the performance between different network structures an average episodic reward is used and is defined as the total reward over an entire episode and averaged over 100 episodes. The graphs presented are only using evaluation episodes, with a deterministic policy. For every 300 training episodes, the policy is evaluated over 300 evaluation episodes and the evaluation metrics are computed. For more details about the evaluation method see [6].

The improvement of using Dropout and Experience replay, from Section V, are clearly shown in Fig. 4 and 5. Studying the red curve in Fig. 4, with all methods included, the best policy had a success rate of 98%, average episodic reward 0.8 and CTR at 40%.

A. Effect of using Experience replay and Dropout

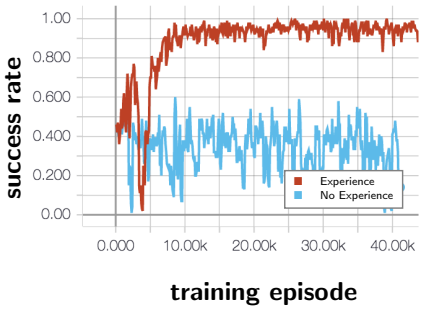

Fig. 4. Success rate trend comparing using experience replay (red) and not using experience replay (blue)

Without either method the success rate does not converge to a value higher than 60%. When experience replay was not used, the highest success rate was 53%, average episodic reward −0.1 and collision to timeout ratio at 90%.

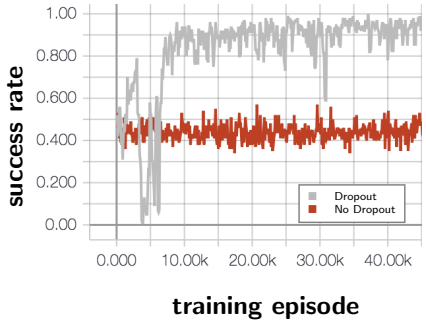

Fig. 5. Success rate trend comparing using dropout (grey) and not using dropout (red)

In the case of not using Dropout, not only was the training time significantly higher, the best policy had a average success rate of 58%, average episodic reward −0.7 and a CTR at 90%. Compared to not using dropout, not using experience replay has a higher variation on the success rate.

B. Comparing DQN and DRQN

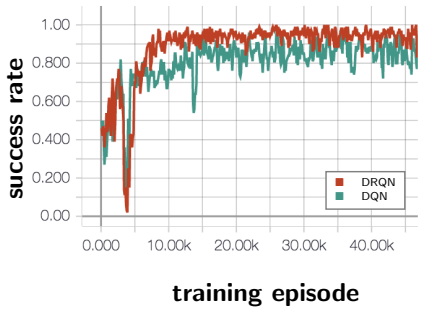

Fig. 6. Graphs comparing the performance of a DRQN (red) and a normal DQN (green). Showing the DRQN outperforming the DQN.

In Fig. 6, we can see the effect off having a recurrent layer, by comparing a DQN without a LSTM layer and with a DRQN with LSTM. The DRQN converges towards a average success rate of 98% with a 0.85% collision rate while the DQN only reached a success rate of 87.5% with a higher collision rate of 1.75%.

C. Effect of sharing weights in the network

Sharing weights for inputs from other cars showed to converge significantly faster for the DRQN compared to having the network train all weights independently of each other, as shown in Fig. 7.

VIII. Conclusion

In this paper, Deep Q-Learning was presented in the domain of autonomous vehicle control. The goal of the ego agent is to drive through an intersection, by adjusting longitudinal acceleration using short-term goals. Short-term

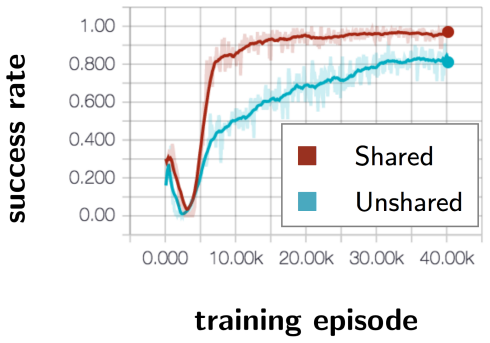

Fig. 7. Showing the affect of using shared weights for observations from other vehicles. The brown curve represents a network using shared weights while the turquoise curve shows a network without shared weights.

goals allowed a smoother and more human-like behavior by controlling the acceleration and comfort with a separate controller. Instead of finding a policy with continuous control output, the problem became a classification problem. This resulted in a policy that is able to respond to other vehicles' actions and behaviors without knowing any traffic rules. The policy learned when it is safe to drive ahead of another vehicle or let them pass, without a prediction model as input, while at the same time consider the comfort of the passenger. The trained policy was able to generalize over different types of driver intentions and varied number of cars.

Results show the importance of using a recurrent layer when the environment is modeled as a POMDP. Meaning, the agent needs multiple observations over time in order to better predict some states, e.g. other vehicles' intentions.

Shared weights between observed vehicles in the first layers showed to improve convergence and performance compared to a fully connected network structure. This means that all observed vehicles are processed the same way independently in which order they are fed to the network and in practice would make it easy to scale number of observed vehicles after training. These results are limited by simulated traffic scenarios and predefined driving behaviors. For future work we plan on implementing this in a real car and traffic scenarios to record other vehicles' behaviors to improve the policy.

The success rate of around 98% is very promising for recognizing behaviors. However, collisions still occur. A collision in this paper is defined by two areas overlapping, and in a real world implementation this does not have to mean an actual collision but instead the safety critical area of car. When the two areas overlap, an intervention from a higher safety critical system would intervene. This way, in the low chances a good action could not be found, the safety of the vehicles can still be guaranteed.

In section III-B, a sliding mode controller was chosen, but this can be replaced by any controller. One other option could be a Model Predictive Controller, where safer actuation can be achieved by using constraints. Also, the actions in this paper used the same controller tuning for all actions, which does not have to be the case. Two action can have the same STG but only differ by the controller's tuning parameters. This way, the agent gains more flexibility while the comfort can maintain intact, possibly increasing the success rate.

References

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to End Learning for Self-Driving Cars. 2016.

[2] Martin Liebner, Michael Baumann, Felix Klanner, and Christoph Stiller. Driver intent inference at urban intersections using the intelligent driver model. In *IEEE Intelligent Vehicles Symposium, Proceedings*, 2012.

[3] Stephanie Lefevre, Christian Laugier, and Javier Ibanez-Guzman. Evaluating risk at road intersections by detecting conflicting intentions. In *IEEE International Conference on Intelligent Robots and Systems*, 2012.

[4] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving.

[5] Sebastian Brechtel, Tobias Gindele, and Rdiger Dillmann. Probabilistic Decision-Making under Uncertainty for Autonomous Driving using Continuous POMDPs.

[6] Robin Grönberg Anton Jansson. Autonomous driving in crossings using reinforcement learning, 2017.

[7] Analysis of an acc system for sliding mode and mpc under transitional manoeuvers. *Mehran University Research Journal of Engineering and Technology*, 31(4), 2012.

[8] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning.

[10] John N Tsitsiklis and Benjamin Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 42(5), 1997.

[11] Long Ji Lin. Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 1992.

[12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. 2012.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[14] Sepp Hochreiter and Jj Urgen Schmidhuber. LONG SHORT-TERM MEMORY. *Neural Computation*, 9(8):1735–1780, 1997.

[15] Guillaume Lample and Devendra Singh Chaplot. Playing FPS Games with Deep Reinforcement Learning.

[16] Hado Van Hasselt, Arthur Guez, Matteo Hessel, Google Deepmind, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude.